APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/837,929	04/19/2001	William J. Walker	500007-A-01-US (Walker)	8223

2292	7590	12/05/2003

BIRCH STEWART KOLASCH & BIRCH
PO BOX 747
FALLS CHURCH, VA 22040-0747

EXAMINER
RUTTEN, JAMES D

ART UNIT	PAPER NUMBER
2122	

DATE MAILED: 12/05/2003 $\mathcal{S}$

Please find below and/or attached an Office communication concerning this application or proceeding.

	Application No.	Applicant(s)
	09/837,929	WALKER, WILLIAM J.
Office Action Summary	Examiner	Art Unit	
	J. Derek Rutten	2122	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE <u>3</u> MONTH(S) FROM
THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed
  after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any
  earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 April 2001*.

2a)☐ This action is **FINAL**.   2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is
   closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-10* is/are pending in the application.

   4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-10* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *28 June 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

   Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

   Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. §§ 119 and 120

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a)☐ All b)☐ Some * c)☐ None of:
   1.☐ Certified copies of the priority documents have been received.
   2.☐ Certified copies of the priority documents have been received in Application No. _____ .
   3.☐ Copies of the certified copies of the priority documents have been received in this National Stage
      application from the International Bureau (PCT Rule 17.2(a)).
   * See the attached detailed Office action for a list of the certified copies not received.

13)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application)
   since a specific reference was included in the first sentence of the specification or in an Application Data Sheet.
   37 CFR 1.78.
   a) ☐ The translation of the foreign language provisional application has been received.

14)☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121 since a specific
   reference was included in the first sentence of the specification or in an Application Data Sheet. 37 CFR 1.78.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____ .
4) ☐ Interview Summary (PTO-413) Paper No(s). _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: .

DETAILED ACTION

1.   Claims 1-10 have been examined.

Specification

2.   The disclosure is objected to because of the following informalities: Pages 5-13 contain

small fonts and single spaced lines which are not in accordance with 37 CFR 1.52(b)(2). Also

see 37 CFR 1.96(b) and MPEP 608.05(a)

        Appropriate correction is required.

3.   The use of the trademark JAVA has been noted in this application. It should be

capitalized wherever it appears and be **accompanied by the generic terminology**.

        Although the use of trademarks is permissible in patent applications, the proprietary

nature of the marks should be respected and every effort made to prevent their use in any manner

which might adversely affect their validity as trademarks.

Claim Rejections - 35 USC § 112

4.   The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the
> subject matter which the applicant regards as his invention.

5.   Claims 1-10 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for

failing to particularly point out and distinctly claim the subject matter which applicant regards as

the invention.

6.   Claim 1 recites the limitation "said JAVA class field" in line 7. There is insufficient

antecedent basis for this limitation in the claim. In the interest of further examination, the

Examiner has interpreted this to mean "a field in said JAVA class".

7.   Claims 1-8 contain the trademark/trade name JAVA. Where a trademark or trade name is

used in a claim as a limitation to identify or describe a particular material or product, the claim

does not comply with the requirements of 35 U.S.C. 112, second paragraph. See *Ex parte*

Simpson, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the trademark or

trade name cannot be used properly to identify any particular material or product. A trademark

or trade name is used to identify a source of goods, and not the goods themselves. Thus, a

trademark or trade name does not identify or describe the goods associated with the trademark or

trade name. In the present case, the trademark/trade name is used to identify/describe an object-

oriented programming language and, accordingly, the identification/description is indefinite.

        Claims 9 and 10 are dependent on claim 8 and suffer the same limitations.

8.   Claims 1, 2, 7, and 8 contain the trademark/trade name XML. Where a trademark or

trade name is used in a claim as a limitation to identify or describe a particular material or

product, the claim does not comply with the requirements of 35 U.S.C. 112, second paragraph.

See *Ex parte Simpson*, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the

trademark or trade name cannot be used properly to identify any particular material or product.

A trademark or trade name is used to identify a source of goods, and not the goods themselves.

Thus, a trademark or trade name does not identify or describe the goods associated with the

trademark or trade name. In the present case, the trademark/trade name is used to

identify/describe a form of the Standard Generalized Markup Language (SGML) called the

"eXtensible Markup Language" and, accordingly, the identification/description is indefinite.

Claims 9 and 10 are dependent on claim 8 and suffer the same limitations.

Claim Rejections - 35 USC § 101

9.   35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

10.   Claims 4-10 are rejected under 35 U.S.C. 101 because the claimed invention is directed

to non-statutory subject matter.

Claims 4-6, and 7-10, merely claimed as an "application programming interface"

and a "data structure", respectively, that is mere arrangements or compilations of facts,

information, or data *per se* and which is merely stored so as to be called "computer-

readable" or even outputted by a computer without creating any functional

interrelationship, either as part of the stored data or as part of the computing processes

performed by the computer ("acts"), then such descriptive material alone does not impart

functionality either to the data as so structured, or to the computer. Thus, such

"descriptive material", non-functional descriptive material, that cannot exhibit any

functional interrelationship with the way in which computing processes are performed

does not constitute a statutory process, machine, manufacture or composition of matter.

And the purely non-functional descriptive material cannot alone provide the practical

application for the manufacture. **Warmerdam**, 33 F.3d at 1361, 31 USPQ2d at 1760. **In**

re Sarkar, 588 F.2d 1330, 1333, 200 USPQ 132, 137 (CCPA 1978). See Examination

Guidelines for Computer-Related Inventions-Final Version, pages 9 & 10. See MPEP §

2106(IV)(B)(1)(b).

Claim Rejections - 35 USC § 102

11.   The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the

basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

12.   Claim 2 is rejected under 35 U.S.C. 102(e) as being anticipated by "Data Binding from

XML to Java, Part 3" by McLaughlin, published September 2000 (hereinafter referred to as

"McLaughlin-3").

McLaughlin-3 discloses:

instantiating an object of the desired JAVA class (See page 3 paragraph 2: "Once

a new instance of the appropriate class is created..."*);*

in the case of said instantiated object, implementing a predefined interface (See

page 3 paragraph 2: "Once a new instance of the appropriate class is created, the mutator

methods are called with the values supplied in the XML document." The mutator

methods are part of a predefined interface for setting attribute values. Listing 3 on page 3

shows the corresponding implementation of a mutator method. Mutator methods are

commonly used for setting attribute values.),

iteratively processing each object included within said instantiated object (See page 4

paragraph 4: "Once all the attributes are read and assigned to the created Java instance,

you need to take each nested element and perform the unmarshalling again.")

according to the steps of:

    retrieving field descriptors associated with an object being processed (See page 4

paragraph 3: "Finally, the nested objects are passed to **accessor methods**, and the top-

level object, which is populated with both member variable values and object **references**,

is **returned** to the calling program." Accessor methods are commonly used to retrieve

attribute values including field descriptors.);

    *creating an object of specified JAVA type for each XML element corresponding to

a field descriptor* (See McLaughlin-3 page 3 paragraph 2: "Once a new instance of the

appropriate class is created, the mutator methods are called with the values supplied in

the XML document."; also page 4 paragraph 4: "Once all the attributes are read and

assigned to the created Java instance, you need to take each nested element and perform

the unmarshalling again."); *and*

    storing the created object in the currently processed object (See page 4 paragraph

4: "Once all the attributes are read and assigned to the created Java instance...").

Claim Rejections - 35 USC § 103

13.   The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all

obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in
> section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are
> such that the subject matter as a whole would have been obvious at the time the invention was made to a person

having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

14.   Claims 1 and 3-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over "Data

Binding from XML to Java, Part 4" by McLaughlin published on October 1, 2000 (hereinafter

referred to as "McLaughlin-4") in view of "Data Binding from XML to Java, Part 3" by

McLaughlin published in September 2000 (hereinafter referred to as "McLaughlin-3").

As per claim 1, McLaughlin-4 discloses:

loading the named JAVA class (See page 2 paragraph 7: "It takes an object and should

return an XML element representation of that object." An object is inherently loaded

from a JAVA class.);

determining if the loaded JAVA class implements a predefined interface (See page 2

paragraph 5: "Any data fields without accessor methods like this will result in that data

being ignored in the process of marshalling." If it is determined that a JAVA class does

not implement a predefined interface including accessor methods, it is ignored.),

said predefined interface comprising annotations including

associating said JAVA class field with a corresponding XML element tag (See

McLaughlin-4 page 3 paragraph 2: "Once the element name is in place, you must obtain

the attributes. Each attribute should be the **name of the field** and the field's value.")

specifying a JAVA class to be instantiated when constructing said JAVA class field from

said XML file (See McLaughlin-4 page 3 paragraph 1: "First, the name of the Java class

is obtained. This name will become the element name of the XML representation being

constructed.")

identifying a JAVA method to invoke for retrieving said JAVA class field (See

McLaughlin-4 page 2 paragraph 5: "...it is expected to have **accessor methods** for all of

its data." Accessor methods are known for retrieving class fields.), *and,*

in the case of said loaded JAVA class implementing said predefined interface

iteratively processing each field descriptor within the loaded JAVA class to retrieve

corresponding XML tag (See McLaughlin-4 page 2 paragraph 7: "It takes an object and

should return an **XML element representation** of that object. If the object contains

references to other Java objects, then you can **recurse**..." As noted earlier, if the accessor

method is not implemented, the descriptor is ignored.)*; and*

transferring field values to new elements created using said corresponding XML tags

(See McLaughlin-4 page 4 paragraph 1: "And once the recursion unwinds, the result is a

complete XML representation of the top-level Java object, which is usable as a root

element in an XML document." Field values are inherently transferred to the resulting

XML representation.).

McLaughlin-4 also discloses the use of parameters in defining the marshalling

function (See page 3, Listing 2 for the definition of "getXMLRepresentation").

McLaughlin-4 does not expressly disclose defining a method with four

parameters, or identifying a JAVA method to invoke for retrieving this method.

However, official notice is taken that the use of parameters in method and

interface definitions is known in the art. An arbitrary number of parameters can be used

in the definition of a method depending on the scope of the data being operated upon, and

the requirements of the method or interface. Parameters are used in method and interface

definitions on occasions where proper data values are not available to the called method.

Further, in an analogous environment, McLaughlin-3 teaches using an interface

including mutator methods which convert data stored in an XML representation of a Java

class instance, into a new Java instance of the same respective class (See page 3

paragraph 2: "Once a new instance of the appropriate class is created, the mutator

methods (all named setXXX) are called with the values supplied in the XML document."

According to McLaughlin's interface, the Java method to invoke for retrieving this

method is standardized based on the name of the attribute.). The mutator method is

inherently identified for retrieval purposes.

It would have been obvious to one of ordinary skill in the art at the time the

invention was made to use the marshalling method of McLaughlin-4 with the derived

mutators of McLaughlin-3 and additional parameters. One of ordinary skill would have

been motivated to provide two-way conversions not only from Java to XML, but also

from XML to Java by using mutators to initialize data members in an object instance.

One would have also been motivated to use parameters to pass data to methods which

would otherwise be out of scope.

As per claim 3, McLaughlin-4 discloses:

annotating said JAVA object to include, for each JAVA object to be converted to

XML, identification of a respective XML tag (See page 3 paragraph 2: "Once the element

name is in place, you must obtain the attributes. Each attribute should be the **name of**

the field and the field's value." The name of Java attributes are inherently annotated

within the Java object.)

identification of a JAVA class to be instantiated when constructing said JAVA object field

from an XML file, (See page 3 paragraph 1: "First, the name of the Java class is obtained.

This name will become the element name of the XML representation being

constructed."),

identification of a JAVA method to invoke for retrieving said JAVA object (See page 2

paragraph 5: "...it is expected to have accessor methods for all of its data." Accessor

methods are known for retrieving objects.)

McLaughlin-4 does not expressly disclose *identification of a JAVA method to*

invoke for retrieving said retrieval method.

However, in an analogous environment, McLaughlin-3 teaches using an interface

including mutator methods which convert data stored in an XML representation of a Java

class instance, into a new Java instance of the same respective class (See page 3

paragraph 2: "Once a new instance of the appropriate class is created, the mutator

methods (all named setXXX) are called with the values supplied in the XML document."

According to McLaughlin's interface, the Java method to invoke for retrieving this

method is standardized based on the name of the attribute.).

It would have been obvious to one of ordinary skill in the art at the time the

invention was made to use the marshalling method of McLaughlin-4 with the derived

mutators of McLaughlin-3. One of ordinary skill would have been motivated to provide

for future unmarshalling of a marshaled object.

As per claim 4, McLaughlin-4 discloses:

a field description retrieval method, for determining JAVA conversion parameters

by examining an annotation associated with each JAVA element to be converted to XML

(See McLaughlin-4 page 2 paragraph 7: "It takes an **object** and should return an XML

element **representation** of that object. If the object contains references to other Java

objects, then you can recurse..." The examination of annotations is inherent when

processing objects.),

said annotation defining for each JAVA element at least a corresponding XML tag, a

corresponding object class (See McLaughlin-4 page 3 paragraph 1: "First, the name of

the Java **class** is obtained. This name will become the **element name** of the XML

representation being constructed."),

a corresponding field retrieval method (See McLaughlin-4 page 2 paragraph 5: "...it is

expected to have **accessor methods** for all of its data." Accessor methods are known for

retrieving objects fields.).

McLaughlin-4 does not expressly disclose *a corresponding method retrieval*

method.

However, in an analogous environment, McLaughlin-3 teaches using an interface

including mutator methods which convert data stored in an XML representation of a Java

class instance, into a new Java instance of the same respective class (See page 3

paragraph 2: "Once a new instance of the appropriate class is created, the mutator

methods (all named setXXX) are called with the values supplied in the XML document."

According to McLaughlin's interface, the Java method to invoke for retrieving this method is standardized based on the name of the attribute.). The mutator method is inherently identified for retrieval purposes.

It would have been obvious to one of ordinary skill in the art at the time the invention was made to use the marshalling method of McLaughlin-4 with the derived mutators of McLaughlin-3. One of ordinary skill would have been motivated to provide two-way conversions not only from Java to XML, but also from XML to Java by using mutators to retrieve methods to initialize data members in an object instance.

As per claim 5, the rejection of claim 4 is incorporated, and McLaughlin further discloses:

loading the named JAVA class (See McLaughlin-4 page 2 paragraph 7: "It takes an object and should return an XML element representation of that object." An object is inherently loaded from a JAVA class.);

determining if the loaded JAVA class implements a predefined interface (See McLaughlin-4 page 2 paragraph 5: "Any data fields without accessor methods like this will result in that data being ignored in the process of marshalling." If it is determined that a JAVA class does not implement accessor methods, it is ignored.),

in the case of said loaded JAVA class implementing said predefined interface iteratively processing each field descriptor within the loaded JAVA class to retrieve corresponding XML tag (See McLaughlin-4 page 2 paragraph 7: "It takes an object and should return an XML element representation of that object. If the object contains

references to other Java objects, then you can recurse..." As noted earlier, if the accessor

method is not implemented, the descriptor is ignored.); *and*

transferring field values to new elements created using said corresponding XML

tags (See McLaughlin-4 page 4 paragraph 1: "And once the recursion unwinds, the result

is a complete XML representation of the top-level Java object, which is usable as a root

element in an XML document.").

McLaughlan-4 does not expressly disclose defining a method with four

parameters.

However, official notice is taken that the use of parameters in method and

interface definitions is known in the art. An arbitrary number of parameters can be used

in the definition of a method depending on the scope of the data being operated upon, and

the requirements of the method or interface. Parameters are used in method and interface

definitions on occasions where proper data values are not available to the called method.

It would have been obvious to one of ordinary skill in the art at the time the

invention was made to use the marshalling method of McLaughlin-4 with additional

parameters. One of ordinary skill would be motivated to provide the marshalling method

with any data that a programmer would want to persist.

As per claim 6, the rejection of claim 4 is incorporated, and McLaughlin-4 further

discloses:

instantiating an object of the desired JAVA class (See McLaughlin-4 page 2

paragraph 7: "It takes an object and should return an XML element representation of that

object." An object is inherently instantiated from a JAVA class);

in the case of said instantiated object implementing a predefined interface (See

McLaughlin-4 page 2 paragraph 5: "Any data fields without accessor methods like this

will result in that data being ignored..."),

iteratively processing each object included within said instantiated object (See

McLaughlin-4 page 2 paragraph 7: "If the object contains references to other Java

objects, then you can recurse...")

according to the steps of:

retrieving field descriptors associated with an object being processed (See

McLaughlin-4 page 3 paragraph 2: "Once the element name is in place, you must obtain

the attributes. Each attribute should be the name of the field and the field's value.");

McLaughlin-4 does not expressly disclose creating a specified Java object for

each XML element, and storing that object.

However, McLaughlin-3 teaches:

creating an object of specified JAVA type for each XML element corresponding to a field

descriptor (See McLaughlin-3 page 3 paragraph 2: "Once a new instance of the

appropriate class is created, the mutator methods are called with the values supplied in

the XML document."; also page 4 paragraph 4: "Once all the attributes are read and

assigned to the created Java instance, you need to take each nested element and perform

the unmarshalling again."); *and*

storing the created object in the currently processed object (See McLaughlin-3 page 4

paragraph 4: "Once all the attributes are read and assigned to the created Java instance).

It would have been obvious to one of ordinary skill in the art at the time the

invention was made to use the object processing of McLaughlin-4 with the XML

translation of McLaughlin-3. One of ordinary skill would have been motivated store an

accurate and complete representation of an object that could easily be sent through a

network and reconstituted at a later time.

As per claim 7, McLaughlin-4 discloses:

associating said JAVA class field with a corresponding XML element tag (See

McLaughlin-4 page 3 paragraph 1: "First, the name of the Java class is obtained. This

name will become the element name of the XML representation being constructed.");

specifying a JAVA class to be instantiated whenconstructing said JAVA class field

from said XML file (See McLaughlin-4 page 3 paragraph 2: "Once the element name is in

place, you must obtain the attributes. Each attribute should be the name of the field and

the field's value.");

identifying a JAVA method to invoke for retrieving said JAVA class field (See

McLaughlin-4 page 2 paragraph 5: "...it is expected to have **accessor methods** for all of

its data." Accessor methods are known for retrieving class fields.).

McLaughlin-4 also discloses the use of parameters in defining the marshalling

function (See page 3, Listing 2 for the definition of "getXMLRepresentation").

McLaughlin-4 does not expressly disclose the use of four parameters, or *identifying a JAVA method to invoke for retrieving this method.*

However, official notice is taken that the use of parameters in method and interface definitions is known in the art. An arbitrary number of parameters can be used in the definition of a method depending on the scope of the data being operated upon, and the requirements of the method or interface. Parameters are used in method and interface definitions on occasions where proper data values are not available to the called method.

Further, in an analogous environment, McLaughlin-3 teaches using an interface including mutator methods which convert data stored in an XML representation of a Java class instance, into a new Java instance of the same respective class (See page 3 paragraph 2: "Once a new instance of the appropriate class is created, the mutator methods (all named setXXX) are called with the values supplied in the XML document." According to McLaughlin's interface, the Java method to invoke for retrieving this method is standardized based on the name of the attribute.).

It would have been obvious to one of ordinary skill in the art at the time the invention was made to use the marshalling method of McLaughlin-4 with the derived mutators of McLaughlin-3 along with multiple parameters. One of ordinary skill would have been motivated to provide two-way conversions not only from Java to XML, but also from XML to Java by using mutators to initialize data members in an object instance. One would have also been motivated to use parameters to pass data to methods which would otherwise be out of scope.

As per claim 8, the rejection of claim 7 is incorporated.

McLaughlin-4 discloses the XML representation of Java collections (See page 2 paragraph 7).

McLaughlin-4 does not expressly disclose *specifying a type of JAVA object to instantiate for an XML element representing a collection.*

However, in an analogous environment, McLaughlin-3 teaches conversion of a supplied string value in XML to a Java type (See Listing 4, also page 4 paragraph 2: "Once this data has been converted to the appropriate type, reflection can be used to invoke the accessor method and pass in the converted data type. This enables all attributes and their values in the XML document to be stored as method variables and values in the resulting Java instance.")

It would have been obvious to one of ordinary skill in the art at the time the invention was made to use the marshalling method of McLaughlin-4, with the type specification of McLaughlin-3. One of ordinary skill would have been motivated to supply type information to an XML representation of a Java class. Strong typing is essential to the Java programming language.

As per claim 9, the rejection of claim 8 is incorporated.

McLaughlin-4 further discloses *specifying a tag name to use for each element representing a collection* (See McLaughlin-4 page 3 paragraph 1: "First, the name of the Java class is obtained. This name will become the element name of the XML representation being constructed.").

15. Claim 10 is rejected under 35 U.S.C. 103(a) as being unpatentable over McLaughlin-4 in view of McLaughlin-3 as applied to claim 8 above, and further in view of "Data Binding from XML to Java Code, Part2" by McLaughlin published in August 2000 (hereinafter referred to as "McLaughlin-2").

McLaughlin-4 does not expressly disclose the use of a hash table.

However, in an analogous environment, McLaughlin-2 teaches the use of a hash table in Java to store multiple interfaces and implementations (See page 2 paragraph 5)

It would have been obvious to one of ordinary skill in the art at the time the invention was made to use the collection of McLaughlin-4 with the hash table of McLaughlin-2. One of ordinary skill would have been motivated to represent Java objects and data in a compact and easily searchable form.

Conclusion

16. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

"Data binding from XML to Java code, Part 1" is the introductory document to the prior art made of record, and outlines some basic concepts of marshalling and unmarshalling Java objects with XML.

U.S. Patent 5,497,491 to Mitchell et al. teaches the use of metadata in an object-oriented environment as a means for data transfer to an external computing environment.

U.S. Patent 5,944,781 to Murray teaches the use of the Java programming language's built-in serialization facilities for storing objects.

"Using Castor XML" by Exoffice Technologies Inc. published in May 2000 gives a brief overview of an XML databinding framework which marshals Java objects to XML.

17.   Any inquiry concerning this communication or earlier communications from the examiner should be directed to J. Derek Rutten whose telephone number is (703) 605-5233. The examiner can normally be reached on M-F 6:30-3:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on (703)305-4552. The fax phone number for the organization where this application or proceeding is assigned is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703)306-5484.

jdr

Chanh C-D m
Primary Patent Examiner
A.Unit : 2122
12/1/03